Proving properties of logic programs: A Progress Report

PROVING PROPERTIES OF LOGIC PROGRAMS: A PROGRESS REPORT

A Bundy, D Sannella, R Desimone,
F Giunchiglia, F van Harmelen,
J Hesketh, P Madden, A Smaill,
A Stevens & L Wallen

DAI RESEARCH PAPER NO. 361

Paper submitted to 1988 Alvey Conference

Proving Properties of Logic Programs: A Progress Report

Alan Bundy, Don Sannella,
Roberto Desimone, Fausto Giunchiglia, Frank van Harmelen,
Jane Hesketh, Pete Madden, Alan Smaill,
Andrew Stevens, and Lincoln Wallen
Department of Artificial Intelligence
University of Edinburgh

January 20, 1988

Abstract

We outline the progress we have made in connection with the Alvey Grant "Proving Properties of Logic Programs" (SERC GR/D/44270 and Alvey IKBS 137). This grant runs for three years from 1st November 1985. The grant holders are Professor Alan Bundy and Dr Don Sannella and it employs or has employed Dr Fausto Giunchiglia, Frank van Harmelen, Jane Hesketh, Dr Alan Smaill and Dr Lincoln Wallen as Research Associates. Pete Madden and Andrew Stevens are attached Ph.D. students.

1 Introduction

It is feared that current software techniques will not be adequate to meet the demands for quantity and complexity being placed on them. One of the fields whose development could play a role in tackling this problem is that of automatic programming, *ie* the automation of (part of) the processes of the generation of programs from specifications; of the verification that a program meets its specification; and of the transformation of inefficient programs into more efficient programs with the same specification. So software reliability could be improved, provided that it is easier to write bug-free specifications than bug-free programs.

Three recent developments are of help in this area.

- logic and functional programming languages, such as Prolog and Miranda;
- the realisation that constructive logic provides a bridge between logical and computational domains (thus relating proofs closely to programs);
- proof guidance techniques from automatic theorem proving.

So using logic programming and constructive logic, the task of generating programs can be treated as the task of proving a theorem, and automatic proof guidance techniques can be used.

Logic Programming

In logic programming languages the specification of a program and source (inefficient) and target (efficient) programs can all be written in the same language, namely a formal logic. So the three processes of synthesis, verification and transformation of programs can be treated in a uniform framework, raising very similar problems. For logic programming, the relation specification/program or program/program according to which transformations are carried out may vary according to which of these problems we are interested in. It is also possible simply to prove that a target program has some useful property, as in [Boyer & Moore 73] and [Boyer & Moore 79].

The advantage of logic programming in this situation over the case where the specification language is different from the target program language is that in the second case optimisation and synthesis cannot be treated uniformly, and in practice the resulting combination of both processes leads to an increase of complexity.

When the specification and implementation languages are the same, it should be easier to write specifications than programs, since programs are written with efficiency considerations in mind whereas such considerations are not involved in writing specifications.

Constructive Logic

Recently increasing attention has been paid to the use of *constructive logic* as providing a link between logical formalism and the computational content embodied in statements in such constructive logics ([Martin-Löf 79, Constable et al 86]).

In particular, we can suppose that we are interested in some relation $\Phi(x, y)$ that we want to hold between input x and output y. Then simply by finding a constructive proof of the statement

$$\forall x \exists y \Phi(x, y)$$

we also find, by a routine computation, an algorithm (the "extract term" τ) with the property that there is a proof that

$$\forall x \Phi(x, \tau(x)).$$

So such a τ computes the output from the input in such a way as to provably satisfy the given specification. In this way, the problem of program synthesis and verification can be treated using the better understood techniques of theorem proving.

It is essential that the logic used should be constructive; for example, when a theorem is proved by considering a number of cases, it is constructively necessary to be able to

tell the cases apart: if we are in such a position, the resulting algorithm will be able to appeal to some test as to which case is relevant. In classical logic, there may be no such test, so a proof that uses (classical) axioms like

$$\phi \vee \neg\phi$$

in a proof by cases, where there is no decision procedure for ϕ, will fail to yield an algorithm in the way described above.

An implementation of this form of program generation and verification has been implemented at Cornell ([Constable et al 86]) based on Martin-Löf's constructive type theory. This is an interactive system, with little to help the user in the search for proofs within the system; this project aims to use ideas on search control, in particular meta-level inference, to facilitate this process.

As is the case for logic programming, this framework can be used in different ways. For example, for straight verification we can conjecture a target program r and show that it satisfies a specification by proving

$$\forall x \Phi(x, r(x)).$$

1.3 Meta-Level Inference

For several years the Mathematical Reasoning Group in the Department of Artificial Intelligence under the direction of Prof A Bundy has investigated the use of meta-level inference as a tool for controlling implementations of mathematical activities, such as solving equations or proving theorems.

In this method, the control and inference for some particular theory, say algebraic equations (called the object theory), is represented in a separate mathematical theory (called the meta-theory) where the formulae, axioms, and inference patterns of the object theory are represented as objects, and so can be reasoned about. The axioms of the meta-theory describe properties of the object theory and so in the meta-theory we can formally reason about solutions to object-level problems and different methods for obtaining these solutions. This method was successfully applied to the area of solving equations in the PRESS program ([Bundy & Welham 79, Sterling et al 82]).

Later the same technique was applied to the domain of logic programs in the IMPRESS program ([Sterling & Bundy 82]); this was capable of verifying Prolog programs, using a proof plan to guide the verification process.

The presence of an explicit meta-level theory also makes it possible to apply deduction and learning notions to the meta-theory itself. The LP program ([Silver 84]) was capable of generalising from a sample solution a plan that could apply in greater generality. So meta-level inference not only helps to guide the object-level inference processes, thus making more complex tasks tractable, it also opens the way to automatic learning of this meta-level knowledge.

2 Goals

The overall goal is to incorporate techniques from meta-level reasoning into a theorem proving program so as to tackle the problems of synthesis and verification of programs, building both on work in logic programming (*eg* [Hogger 81]) and in the relation of programs to proofs via constructive logic (*eg* [Constable 82]). Using the notion of proof plan, the lower-level tactics used to guide object-level inference can be combined so as to guide the search for proofs of certain forms. It is possible to prove properties of these proof plans, provided that they are expressed in an appropriate declarative language.

Other goals included the automatic transformation of programs treated via the transformation of proofs, and the exploration of automatic learning of proof plans from sample proofs.

The system in use is a customised version of Cornell University's NuPRL system [Constable et al 86], which is an implementation of Martin-Löf's Type Theory. As a logical system, this is a rich language incorporating a higher order constructive logic; as a programming language it corresponds to a polymorphic higher-order functional language such as ML or Miranda.

The meta-level tactics (as in [Gordon et al 79]) which implement the proof construction operations are written in ML; the typing of the ML language is used to ensure the correctness of these proof construction operators, making it impossible to produce an unsound proof.

3 Progress and Current Status

The original Cornell system was first ported to the SUN workstation and adapted to run under Xwindows. It has been adapted in several ways to make it more suitable for its projected role.

A library module system has been added, allowing not only hierarchical treatment of sets of definitions and theorems built up inside the system, but also allowing "abstract" theories to be built up, where a number of objects and properties can be assumed, and the development carried ahead under these assumptions, to be fully justified later. This allows the incorporation of a degree of top-down implementation (in the usual manner of specification refinement) into the basic bottom-up approach, and makes the use of the system for life-size problems more practicable.

The type system has been extended with the addition of a new induction principle (for "course-of-values" induction). Since inductive proofs play a central role in this work (showing that a specification can be satisfied by some recursively defined object corresponds to showing termination of an algorithm), the provision of this more flexible form allows the investigation of alternative inductive proofs and comparisons between their associated algorithms.

Using the system, problems such as the synthesis of a variety of sorting algorithms and the synthesis and verification of a unification algorithm have been carried out.

These sorting algorithms are being used as a domain to examine how the transformation of proofs can be used to optimise program efficiency ([Madden 87]). A proof from which a program is derived using the programs-as-proofs principle will contain more information than the program derived from it. Using notions from analogy, proofs may be systematically related in such a way that more efficient programs can be obtained by transforming the associated proofs. The crucial element in the proof is which forms of induction are invoked and how they are related.

In order to exploit the implicit proof-plan found in Boyer and Moore's theorem prover, it is important to have an explicit representation in which proof plans can be specified and tactics combined. Work is currently in hand on elaborating such a specification language. The basic proof plan for inductive proofs has been elaborated in several formalisms to compare their merits, and proofs of the correctness of such plans have been set out [Bundy 88].

Ultimately plans should be capable of reacting to local failures by means of dynamic patching.The plans could themselves be the object of systematic transformation. We are also exploring the relation between these proof plans and the AI planning tradition.

The use of pre-condition analysis to learn proof plans has been investigated, resulting in extensions to the previous techniques ([Desimone 87]) and applied to NuPRL proofs corresponding to insert functions in lists.

Currently investigation is also proceeding in the use of NuPRL itself as a possible specification language, using some form of reflection as in [Knoblock & Constable 86]. This would allow the processes of specification, synthesis and verification of tactics to be treated in the same way as their object-level counterparts. Building on the work of Weyrauch and the FOL group ([Weyhrauch 80]), this would allow formalised reasoning at both the meta-level and object-level to take place in a single system, providing a formal link between meta-reasoning and object-level reasoning. This is one systematic way in which the user of the system can extend its metamathematical capability in the same way as is already possible at the object-level using the definition and library mechanisms.

4 Summary

We outline the work undertaken in connection with the Alvey "Proving Properties of Logic Programs" Grant. The project aims to synthesise experience in automated theorem proving (in particular Boyer and Moore) and the proofs-as-programs school (using constructive logic, in particular the NuPRL system) to tackle the problems of the automated synthesis, verification and transformation of programs. The NuPRL system has been adapted and altered to make it more suitable for this task, and the experience of the group in meta-level inference brought to bear on the problems raised. The notion of proof-plan has proved central in allowing AI techniques of learning, analogy, and explicit meta-level inference to be applied. It seems that the constructive logic approach opens up many possible avenues of research, though the problem of automating significant applications remains difficult.

References

[Boyer & Moore 73] R.S. Boyer and J.S. Moore. Proving theorems about lisp functions. In N. Nilsson, editor, *Proceedings of the third IJCAI*, pages 486–493, International Joint Conference on Artificial Intelligence, August 1973. Also available from Edinburgh as DCL memo No. 60.

[Boyer & Moore 79] R.S. Boyer and J.S. Moore. *A Computational Logic.* Academic Press, 1979. ACM monograph series.

[Bundy & Welham 79] A. Bundy and B. Welham. *Using meta-level descriptions for selective application of multiple rewrite rules in algebraic manipulation.* Working Paper 55, Dept. of Artificial Intelligence, Edinburgh, May 1979. Revised version in Artificial Intelligence 16(2). Also Research Paper 121.

[Bundy 88] A. Bundy. *The Use of Explicit Plans to Guide Inductive Proofs.* Research Paper forthcoming, Dept. of Artificial Intelligence, Edinburgh, 1988. Short version submitted to CADE9.

[Constable 82] R.L. Constable. *Programs as Proofs.* Technical Report TR 82-532, Dept. of Computer Science, Cornell University, November 1982.

[Constable et al 86] R.L. Constable, S.F. Allen, H.M. Bromley, et al. *Implementing Mathematics with the Nuprl Proof Development System.* Prentice Hall, 1986.

[Desimone 87] R.V. Desimone. Explanation-based learning of proof plans. In Y. Kodratoff, editor, *Machine and Human Learning*, Ellis Horwood, 1987. Also available as DAI Research Paper 304. Previous version in proceedings of EWSL-86.

[Gordon et al 79] M.J. Gordon, A.J. Milner, and C.P. Wadsworth. *Edinburgh LCF - A mechanised logic of computation.* Volume 78 of *Lecture Notes in Computer Science*, Springer Verlag, 1979.

[Hogger 81] C.J. Hogger. Derivation of logic programs. *JACM*, 28(2):372–392, April 1981.

[Knoblock & Constable 86] T. B. Knoblock and R.L. Constable. Formalized metareasoning in type theory. In *Proceedings of LICS*, pages 237–248, IEEE, 1986.

[Madden 87] P. Madden. *The Application of Analogy to Program Optimization.* Discussion paper 51, DAI, September 1987.

[Martin-Löf 79] Per Martin-Löf. Constructive mathematics and computer programming. In *6th International Congress for Logic, Methodology and Philosophy of Science*, pages 153–175, Hannover, August 1979. Published by North Holland, Amsterdam. 1982.

[Silver 84] B. Silver. Precondition analysis: learning control information. In *Machine Learning 2*, Tioga Publishing Company, 1984.

[Sterling & Bundy 82] L. Sterling and A. Bundy. Meta-level inference and program verification. In D.W. Loveland, editor, *6th Conference on Automated Deduction*, pages 144–150, Springer Verlag, 1982. Lecture Notes in Computer Science No. 138. Also available from Edinburgh as Research Paper 168.

[Sterling et al 82] L. Sterling, A. Bundy, L. Byrd, R. O'Keefe, and B. Silver. Solving symbolic equations with press. In J. Calmet, editor, *Computer Algebra, Lecture Notes in Computer Science No. 144.*, pages 109–116, Springer Verlag, 1982. Also available from Edinburgh as Research Paper 171.

[Weyhrauch 80] R.W. Weyhrauch. Prolegomena to a theory of mechanized formal reasoning. *Artificial Intelligence*, 13:133–170, 1980.